International Journal of Electrical and Computer Engineering (IJECE)
Vol. 5, No. 1, February 2015, pp. 55~63
ISSN: 2088-8708

An Improved Design of Linear Congruential Generator based on Wordlengths Reduction Technique into FPGA

Hubbul Walidainy, Zulfikar
Department of Electrical Engineering, Syiah Kuala University

Article Info

Article history:

Received Oct 22, 2014
Revised Dec 12, 2014
Accepted Dec 30, 2014

Keyword:

FPGA
Improve LCG
Linear Congruential Generator
Wordlengths reduction
Xilinx

ABSTRACT

This paper exposes an improved design of linear congruential generator (LCG) based on wordlengths reduction technique into FPGA. The circuit is derived from LCG algorithm proposed by Lehmer and the previous design. The wordlengths reduction technique has been developed more in order to simplify further circuit. The proposed design based on the fact that in applications only specific input data were used. Some nets connections between blocks of the circuit are ignored or truncated. Simulations either behavior or timing have been done and the results is similar to its algorithm. Four best Xilinx chips have been chosen to extract comparison data of speed and occupied area. Further comparison of occupied area in terms of flip-flop and full adder has been made.In general, the proposed design overcome the previous published LCG circuit.

Corresponding Author:

Hubbul Walidainy
Departement of Electrical Engineering, Syiah Kuala University,
Jl Syech Abdur Rauf, Darussalam, Banda Aceh, Indonesia.
Email: hwalidainy@unsyiah.ac.id

1. INTRODUCTION

The use of random numbers has become habit in daily activities since long times ago. Right now, even a cheap kid's toy contains a random number circuit inside it. For instance, in a toy that mimic mobile phone will ring variation types of sound when the same button is pressed many times.

Random numbers theory has been re-introduced in the last several decades. Linear congruential generator (LCG) that introduced 1954 by Lehmer [1] is the oldest and the most commonly used pseudorandom number generator (PNG) [2]. Park and Miller suggested good parameters for LCG [3]. Their idea has been used in Matlab for generating uniform random numbers until now [4].

Many other theories of random number generators have been proposed and also used in many applications. Blum Blum Shub, Wichmann-Hill, Complementary multiply with carry, Inversive congruential generator, ISAAC (cipher), Lagged Fibonacci generator, Linear feedback shift register, Maximal periodic reciprocals, Mersenne twister, Multiply-with-carry, Naor-Reingold Pseudorandom Function, RC4 PRGA, Well Equidistributed Long-period Linear, and Xorshift are some of the common and well-known methods [5]-[8].

Hardware for generating random number is also available as well as their algorithms. Hardware has been used since 2008. LETech is the fastest among all hardware for generating random numbers, this hardware has been developed and marketed since 2008 [9], [10].

Research for searching suitable algorithms of generating random numbers is well establishing field until now. Researchers use field programmable gate arrays (FPGA) for testing their algorithms. Some of good design has been realized into hardware and sold into the market [5], [10].

Initially, The algorithm of LCG that has been combined with Monte Carlo method used for generating non uniform random number in Matlab [11]. Later, the development has been implemented in FPGA [12]. In the paper, the increment factor (c) has been ignored (c=0). Later then, the technique for

generating random number based on full LCG algorithm also available [13]. The paper exposed the LCG circuit design without ignoring the increment factor. We analyze that the technique may be improved further.

In this work, we design the more efficient circuit by reducing wordlengths of some input data. Moreover the technique proposed in [13] will produce slightly different random numbers than the original LCG algorithms.

The rest of the paperis organized as follows. Section 2 deals with theory of LCG algorithm. The design of improved LCG circuit for FPGA implementation and nets modifications are explained in section 3. The deep analysis and implementations are presented insection 4. Finally, the conclusions are summarized in section 5.

2. THEORY

2.1. Linear Congruential Generator

There is a popular method and most used to generate random number called linear congruential generator. The idea was introduced by Lehmer according to sequential formula in (1) [1].

$$X_{n+1} = (aX_n + c) \bmod m \quad (1)$$

Where m is *modulus*, $a$is *multiplier*, c is *increment*. Parameters a, c and m have to be chosen carefully in order to avoid repetition of similar numbers before m [6]-[8]. Park & Miller suggested a good results will be obtained by choosing c=0 [3].

The *modulusm* should be a large prime integer, *multiplier* a will be an integer in the range 2, 3, . . . , m-1. The cycle length of LCG will never exceed *modulus m*, but it can be maximized using three following conditions [7], [14]:

- c is relatively prime to *modulusm*,
- *multipliera*-1 is a multiple of every dividing *modulusm*,
- *multipliera*-1 is a multiple of four when *modulusm* is *a* multiple of four.

2.2. Parameters in Common Use of LCG

The requirements mentioned in the previous section are referred to Hull-Dobell theorem [15]. LCG are able to produce the pseudorandom number that can pass test of randomness. The condition is sensitive in choosing the good parameters *c, m*, and *a*.

In history, poor choices had been led to the ineffective realizations or implementations of LCG itself. As an example of the poor parameters choice is RANDU (see Table 1), this method was commonly implemented in the early 1970 and cause to many results that are currently being questioned.

Random numbers resulted by LCG will be more efficientwhen the modulus approach a very high numbers, often reach the maximum computer (machine) ability such as m=2^{32} or m=2^{64}. Table 1 lists the parameters of LCGs in commonly use, including built in functions such as *rand()* that used in runtime libraries of various compilers [2].

Table 1. Parameters in common used for LCG applications [2]

Source	m	(multiplier) a	(increment) c
Numerical Recipes	2^{32}	1664525	1013904223
Borland C/C++	2^{32}	22695477	1
glibc (used by GCC)	2^{31}	1103515245	12345
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++	2^{31}	1103515245	12345
Borland Delphi, Virtual Pascal	2^{32}	134775813	1
Microsoft Visual/Quick C/C++	2^{32}	214013 ($343FD_{16}$)	2531011 ($269EC3_{16}$)
Microsoft Visual Basic (6 and earlier)	2^{24}	1140671485 ($43FD43FD_{16}$)	12820163 ($C39EC3_{16}$)
RtlUniform from Native API	$2^{31} - 1$	2147483629 ($7FFFFFED_{16}$)	2147483587 ($7FFFFFC3_{16}$)
Apple CarbonLib, C++11's minstd_rand0	$2^{31} - 1$	16807	0
C++11's minstd_rand	$2^{31} - 1$	48271	0
MMIX by Donald Knuth	2^{64}	6364136223846793005	1442695040888963407
Newlib	2^{64}	6364136223846793005	1
VAX's **MTH$RANDOM**, old versions of glibc	2^{32}	69069	1
Java's java.util.Random, glibc [ld]rand48[_r]()	2^{48}	25214903917	11
RANDU	2^{31}	65539	0

3. LCG CIRCUIT DESIGN

3.1. General Circuit of LCG

Figure 1 show the LCG circuit proposed in [13]. The designed circuit used equal wordlengths of n for connection between blocks. The designed circuit consist of (assumed modulus=2^n):

- One n x n-bit multiplier
- One n-bit 2-to-1 multiplexer
- One n-bit adder
- 3 x n enable buffers (B_1, B_2, B_3)
- n buffers (B_4)

For certain applications, the design may be improved further. By using the same circuit blocks, we proposed a circuit for a more efficient area. This design requires some assumptions.

The design based on the fact that in application only specific multiplier and increment were used [2]. Here as an example, we design an efficient circuit for *modulus* of maximum 8-bit.It is assumed that the wordlength of *multiplier* use 3-bit and the wordlength of *incrementc* use 2-bit data. Figure 2 show a slight modification circuit of the previous design.

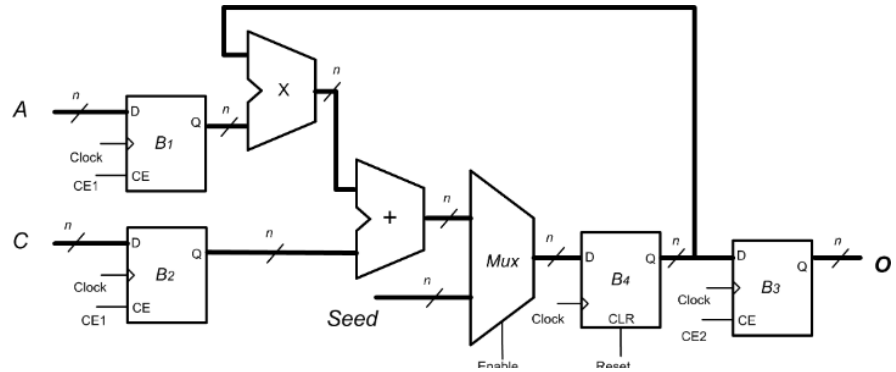

Figure 1. General circuit of linear congruential generator [13]

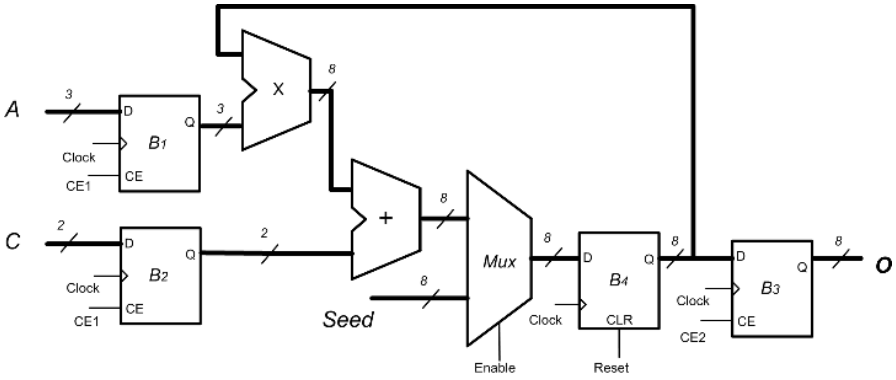

Figure 2. Proposed circuit design of a more efficient area (n=8)

The proposed circuit in the Figure 2 is also controlled by two signals *enable* and *reset*. The controlling process is equal to the previous design. Initially, signal *reset* have to be HIGH (*enable*=LOW) in order to clear the stored values in the buffer B_4. Signal *enable* determine whenever the operation should be started. Pre-defineddataof *seed*, *increment* and *multiplier* have to be available at the input ports just before signal *enable* goes HIGH (*reset*must be LOW). Afterthat, each times the clock goes HIGH, a random number is produced. Figure 3 views the circuit configuration of the two controlsignals.

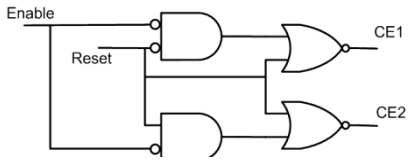

Figure 3. Signal controls of the designed LCG circuit [13]

In practice, the proposed circuit will also reduce number of net connections between CE_1, CE_2 and buffers.

3.2. Wordlengths Reduction

The circuit of Figure 2 used equal wordlengths everywhere. Therefore, the multiplier block have to be implemented with an 8x8-bit circuit (assumed n=8). However, the proposed design requires smaller circuit. Figure 4 shows the net connections configuration of the multiplier block.

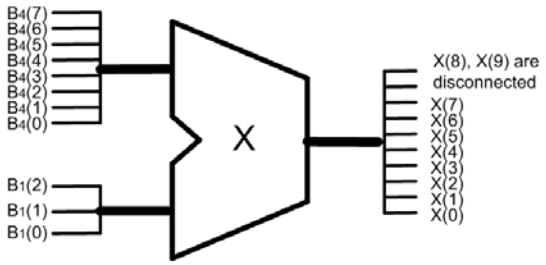

Figure 4. Proposed wordlengths reduction in multiplier's block

Based on arithmetic rules and to reduce area [16], the multiplication of B_4 (8 bit) and B_1 (3 bit) would require 10 bit at the output. In the design, we simply truncated (disconnected) the twohigher nets of X(8) and X(9). Similarly, we can do the same thing to the nets of the adder. Again, based on arithmetic rules and to reduce area [15], the addition of X (8 bit) and B_2 (2 bit) would require 9 bit at the output. But, in this case only one net (M(8)) disconnected as shown in the Figure 5.

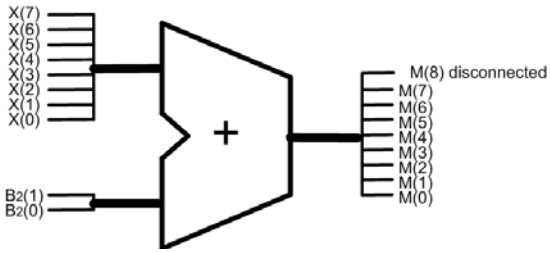

Figure 5. Proposed wordlengths reduction in adder's block

4. SIMULATION AND COMPARISONS

In order to evaluate whether the design works properly, we simulate the proposed circuit in Figure 2. Both behavior and timing simulation have been done using Xilinx ISE Design Suite 14.2.Some important information of synthesis result is presented. The comparison to the previous design of area and speed have been done to several Xilinx's chips.

The implementations have been done using three wordlengths 8-bit, 16-bit, and 31-bit. The wordlengths for *seed* and output are equal to wordlengths design. Meanwhile, some input use smaller wordlengths which are multiplier 3-bit, increment 2-bit.

4.1. Behavior Simulation

Figure 6 shows input data and results of behavior simulation using *modulusm* = 2^8, *seed* = 7, *multiplier a* = 3 and *increment c* = 1. It can be seen that the result numbers are random starting from 7 and all numbers are smaller than 2^8.

Figures 7 and 8 show behavior simulation results of *modulusm* = 2^{16} and *m* = 2^{31}, respectively. The simulations also produce numbers which never exceed the *modulus*.

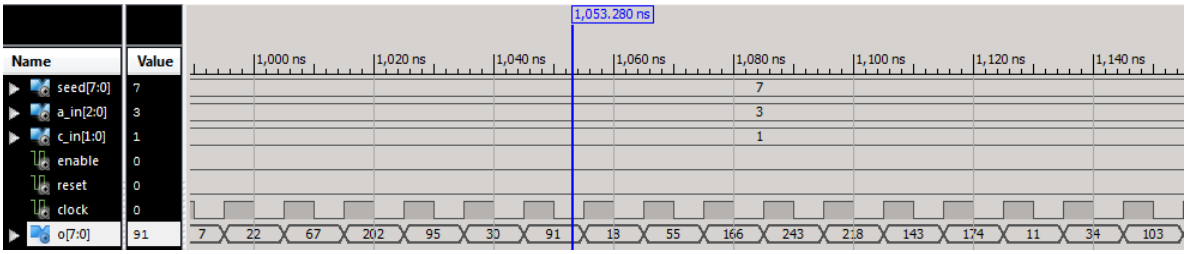

Figure 7. Simulation behavior result of proposed design using *m*=2^{16}, *seed*=7, *a*=3, *c*=1

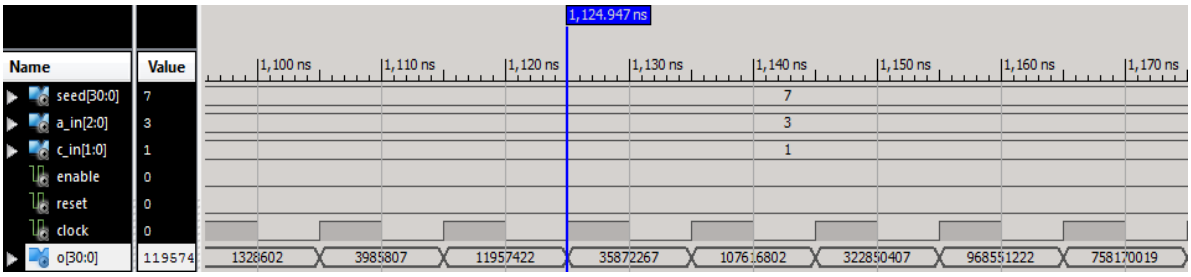

Figure 8. Simulation behavior result of proposed design using *m*=2^{31}, *seed*=7, *a*=3, *c*=1

4.2. Synthesis Results

Some important data after synthesis step of the proposed design circuit using *modulusm*=2^8 into Xilinx Zynq chip are:

```
HDL Synthesis Report

Macro Statistics
# Multipliers                          : 1
 8x3-bit multiplier                    : 1
# Adders/Subtractors                   : 1
 11-bit adder                          : 1
# Registers                            : 4
 2-bit register                        : 1
 3-bit register                        : 1
 8-bit register                        : 2
# Multiplexers                         : 1
 8-bit 2-to-1 multiplexer              : 1

=========================================
Advanced HDL Synthesis Report

Macro Statistics MACs                 : 1
 8x3-to-8-bit MAC                     : 1
# Registers                           : 19
 Flip-Flops                           : 19
# Multiplexers                        : 1
 8-bit 2-to-1 multiplexer             : 1

Device utilization summary:

Selected Device : 7z010clg400-3
Slice Logic Utilization:
 Number of Slice Registers:    19  out of  35200 0%
 Number of Slice LUTs:         30  out of  17600  0%
    Number used as Logic:      30  out of  17600  0%

Slice Logic Distribution:
```

```
   Number of LUT Flip Flop pairs used:40

Number with an unused Flip Flop: 21 out of4052%
   Number with an unused LUT:        10 out of 40 25%
   Number of fully used LUT-FF pairs:9  out of 40 22%
   Number of unique control sets:    3

IO Utilization:
 Number of IOs:               24
 Number of bonded IOBs:       21  out of 10021%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:    1  out of  323%

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 2.436ns (Maximum Frequency: 410.526MHz)
   Minimum input arrival time before clock: 0.892ns
   Maximum output required time after clock: 0.511ns
   Maximum combinational path delay: No path found
```

From HDL synthesis report, the proposed design circuit requires an 8x3-bit multiplier, 11-bit adder, two 8-bit register, one 3-bit register, one 2-bit register and 8-bit 2-to-1 multiplexer. In terms of slice logic utilization, the design occupied 19 slice registers and 30 slice LUTs. The distribution of slice logic from total amount of 40 are 9 for fully used LUT-FF pairs, 10 to unused LUT and 21 for unused flip-flop. The circuit also requires 3 unique control sets.

The maximum frequency of the circuit limited to around 410.52 MHz when it is implemented into Zynq. The minimum input arrival time before clock is 0.892 ns. This means the data should be available (arrive) at input port before that time. The maximum output required time after clock is 0.511 ns.

4.3. Timing Simulation

Figure 9 shows a close view of timing simulation result. The figure show transition between 202 and 95. There are some glitches appears because of the time from clock edge to pads varies. The variation values are ranging from 8.390 ns to 8.527 ns (post-PAR static timing report).

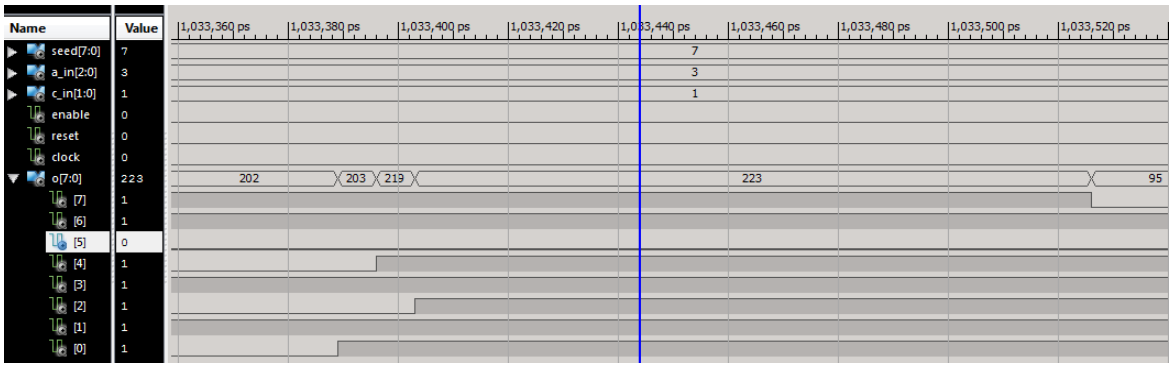

Figure 9. A close look of timing simulation

```
post-PAR static timing report
Clock Clock to Pad
 ------------+-----------------+------------+-----------------+------------+------------------+--------+
             |Max (slowest) clk|  Process   |Min (fastest) clk|  Process   |                  | Clock  |
 Destination |  (edge) to PAD  |   Corner   |  (edge) to PAD  |   Corner   |Internal Clock(s) | Phase  |
 ------------+-----------------+------------+-----------------+------------+------------------+--------+
 O<0>        |         8.390(R)|      SLOW  |         4.033(R)|      FAST  |Clock_BUFGP       |   0.000|
 O<1>        |         8.391(R)|      SLOW  |         4.035(R)|      FAST  |Clock_BUFGP       |   0.000|
 O<2>        |         8.404(R)|      SLOW  |         4.046(R)|      FAST  |Clock_BUFGP       |   0.000|
 O<3>        |         8.440(R)|      SLOW  |         4.071(R)|      FAST  |Clock_BUFGP       |   0.000|
 O<4>        |         8.397(R)|      SLOW  |         4.037(R)|      FAST  |Clock_BUFGP       |   0.000|
 O<5>        |         8.422(R)|      SLOW  |         4.049(R)|      FAST  |Clock_BUFGP       |   0.000|
 O<6>        |         8.425(R)|      SLOW  |         4.052(R)|      FAST  |Clock_BUFGP       |   0.000|
 O<7>        |         8.527(R)|      SLOW  |         4.155(R)|      FAST  |Clock_BUFGP       |   0.000|
 ------------+-----------------+------------+-----------------+------------+------------------+--------+
```

4.4. Comparisons

Four Xilinx chips have been chosen for area and speed comparison between the proposed design circuit and the previous one. Table 2 viewscomparison of occupied area of*modulus* $m=2^8$ (8 bit), $m=2^{16}$ (16 bit) and $m=2^{31}$(31 bit) over Virtex 7, Spartan 6, Kintex 7 and Zynq chips.

Table 2. Occupied area comparison among Xilinx chips

Chips	Area Occupies					
	8 bit (Slices/LUTs)		*16 bit (Slices/LUTs)*		*31 bit (Slices/LUTs)*	
	[13]	*Proposed*	*[13]*	*Proposed*	*[13]*	*Proposed*
Virtex 7	16/10	19/30	32/18	36/62	92/63	66/122
Spartan 6	16/10	19/30	32/18	33/19	92/63	67/122
Kintex 7	16/10	19/30	32/18	35/62	92/63	65/122
Zynq	16/10	19/30	32/18	35/62	92/63	65/122

Based on synthesis report, for $m=2^8$, the required area (both slices and LUTs) of the proposed design is more than the previous design. This is slightly different for $m=2^{16}$, the number of slices is about three times and the number of LUTs is almost equal. When the proposed design is implemented using $m=2^{31}$, the number of slices is less, even though the number of LUTs is still more. Table 2 shows this phenomena.

Table 3 views the speed comparison of the proposed design and the previous one. It can be seen that the proposed design is faster. The maximum frequency that can be reached varies from 154 MHz to 411 MHz. As the wordlengths increases, the maximum frequency decreases. Spartan 6 is the slowest chip, Kintex 7 and Zynq are the best choice.

Table 3. Maximum frequency comparison among Xilinxchips

Chips	Maximum Frequency (MHz)					
	8 bit		*16 bit*		*31 bit*	
	[13]	*Proposed*	*[13]*	*Proposed*	*[13]*	*Proposed*
Virtex 7	270	376	270	361	139	337
Spartan 6	154	248	154	158	73	209
Kintex 7	309	411	270	397	158	369
Zynq	272	411	272	397	140	369

In order to make a more detail comparison, the area of the designed circuit and the previous one arere-implemented,analyzed andwe put some calculations of the synthesis results of the proposed and the previous methods. Table 4 up to Table 9 describe more about area comparison. The area is represented in terms of usedflip-flop and full adder.

From Tables 4 and 5, the numbers of flip-flip and adder of the proposed design is about a half of the previousone. The proposed design become more and more efficient for higher modulus as can be seen inTables 6 and 7 for $m=2^{16}$ and Tables 8 and 9 for $m=2^{31}$.

Table 4. Calculation of area based on synthesis results for *modulus* 8 bit [13]

Circuits	Bit Size	Counts	Flip-Flops	Full Adders
Multipliers	8x8-bit	1	-	64
Adders	16-bit	1	-	16
Registers	8-bit	4	32	-
Total			32	80

Table 5. Calculation of area based on synthesis results for *modulus* 8 bit (proposed design)

Circuits	Bit Size	Counts	Flip-Flops	Full Adders
Multipliers	8x3-bit	1	-	24
Adders	11-bit	1	-	11
Registers	8-bit	2	16	-
	3-bit	1	3	-
	2-bit	1	2	-
Total			21	35

Table 6. Calculation of area based on synthesis results for *modulus* 16 bit [13]

Circuits	Bit Size	Counts	Flip-Flops	Full Adders
Multipliers	16x16-bit	1		256
Adders	32-bit	1	-	32
Registers	16-bit	4	64	
	Total		64	288

Table 7. Calculation of area based on synthesis results for *modulus* 16 bit (proposed design)

Circuits	Bit Size	Counts	Flip-Flops	Full Adders
Multipliers	17x3-bit	1	-	51
Adders	20-bit	1	-	20
Registers	17-bit	1	17	-
	16-bit	1	16	-
	3-bit	1	3	-
	2-bit	1	2	-
	Total		38	71

Table 8. Calculation of area based on synthesis results for *modulus* 31 bit [13]

Circuits	Bit Size	Counts	Flip-Flops	Full Adders
Multipliers	31x31-bit	1	-	961
Adders	32-bit	1	-	32
Registers	31-bit	4	124	
	Total		124	983

Table 9. Calculation of area based on synthesis results for *modulus* 31 bit (proposed design)

Circuits	Bit Size	Counts	Flip-Flops	Full Adders
Multipliers	31x3-bit	1	-	93
Adders	32-bit	1	-	32
Registers	31-bit	2	64	-
	3-bit	1	3	-
	2-bit	1	2	-
	Total		69	125

The proposed design is derivated from the habit of input data, it might be varied based on LCG application. For sure, one thing that can be learned from the design is there are space to reduceoccupied area and improve the speed whatever application it is.

5. CONCLUSIONS

An improved design and implementation of linear congruential generator into FPGA have been done successfully. It is assumed that the designed circuit is used for specific applications. In general, the proposed design circuit is far faster and less in used flip-flop and full adder. In term of slices and LUTs based on FPGA synthesis, the proposed design requires more than the design published in [13]. It can be justified, the previous one used equal wordlengths while the proposed circuit implement various wordlengths. Therefore, the number of slices and LUTs increases for the proposed design.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the financial support from Syiah Kuala University, Ministry of Education and Culture, Indonesia under project Hibah Bersaing, No. 498/UN11/S/LK-BOPT/2014, 26 May 2014.

REFERENCES

[1] D.H. Lehmer, "Random number generation on the BRL high speed computing machines", by M. L. Juncosa. *Math.* Rev. *15* (1954), 559
[2] http://en.wikipedia.org (2014) - Linear congruential generator, 10th March http://en.wikipedia.org/wiki/Linear_congruential_generator
[3] S.K. Park, and K.W. Miller, "Random number generators: good ones are hard to finnd", *Association for Computing Machinery*, 31(10*)*, pp: 1192-2001, 1988.

[4] [Numerical Computing with MATLAB, By Cleve B. Moler, *SIAM*,2008.
[5] http://en.wikipedia.org (2014) - List of random number generators, 11th March http://en.wikipedia.org/wiki/List_of_random_number_generators
[6] N. Harald, "Random Number Generation and Quasi-Monte Carlo Methods", *Society for lndustrial and Applied Mathematics*, Philadelphia, 1992.
[7] A note on random number generation, Christophe Dutang and Diethelm Wuertz, September 2009
[8] *Wolfram Mathematica* ® Tutorial Collection, RANDOM NUMBER GENERATION, 2008
[9] http://www.letech.jpn.com (2014) - Genuine Random Number Generator (GRANG), 10th March http://www.letech.jpn.com/rng/about_rng_e.html
[10] http://en.wikipedia.org (2014) - Comparison of hardware random number generators, 10th March http://en.wikipedia.org/wiki/Comparison_of_hardware_random_number_generators
[11] Zulfikar, "Generating Non Uniform Random Numbers Using Residue and Rejection Methods", *Transaction of Journal Rekayasa Elektrika*, Vol. 8 No. 2, October 2009
[12] Zulfikar, "FPGA Implementations of Uniform Random Number based on Residue Method", *Transaction of Journal Rekayasa Elektrika*, Vol. 11 No. 1, April 2014
[13] Zulfikar and Hubbul Walidaiy, "Design of Linear Congruential Generator based on Wordlengths Reduction Technique into FPGA", *Transaction of International Journal of Electronics Communication Computer Engineering*, Vol. 5 No. 4, pp: 809-813, July 2014
[14] D.E. Knuth, "The Art of Computer Programming: seminumerical algorithms", Vol. 2,3rd edition edn, *Massachusetts: Addison-Wesley*, 2002
[15] F. Severance,"System Modeling and Simulation", *John Wiley & Sons, Ltd*. p. 86,2001
[16] Zulfikar, *et al.*, "FPGA Based Complete Set of Walsh and Inverse Walsh Transforms for Signal Processing", *Transaction of Electronics and Electrical Engineering Journal*, Vol.18. No. 8, Pp: 3-8, October 2012

BIOGRAPHIES OF AUTHORS

Hubbul Walidainy. He was born in Banda Aceh, Aceh, Indonesia, in 1973. He graduated from Electrical Engineering Department at Gadjah Mada University, Yogyakarta, Indonesia, in 1998. The Master Degree in Electrical Engineering from Gadjah Mada University, Yogyakarta, Indonesia, in 2003.
He joined in the Department of Electrical Engineering, Syiah Kuala University, Aceh, Indonesia in 2000, as a teaching staff. His current position is the head of Telecommunication Laboratory.

Zulfikar. He was born in Beureunuen, Aceh, Indonesia, in 1975. He received his B.Sc. degree in Electrical Engineering from North Sumatera University, Medan, Indonesia, the M. Sc. Degree in Electrical Engineering from King Saud University, Riyadh, Saudi Arabia, in 1999 and 2011, respectively
He joined as teaching staff in the Department of Electronics at Politeknik Caltex Riau, Pekanbaru, Indonesia in 2003. He served as head of Industrial Control Laboratory, Politeknik Caltex Riau from 2003 to 2006. In 2006, he joined the Electrical Engineering Department, Syiah Kuala University. His current position is head of Digital Laboratory, and his current research interests include VLSI design and System on Chips (SoC).